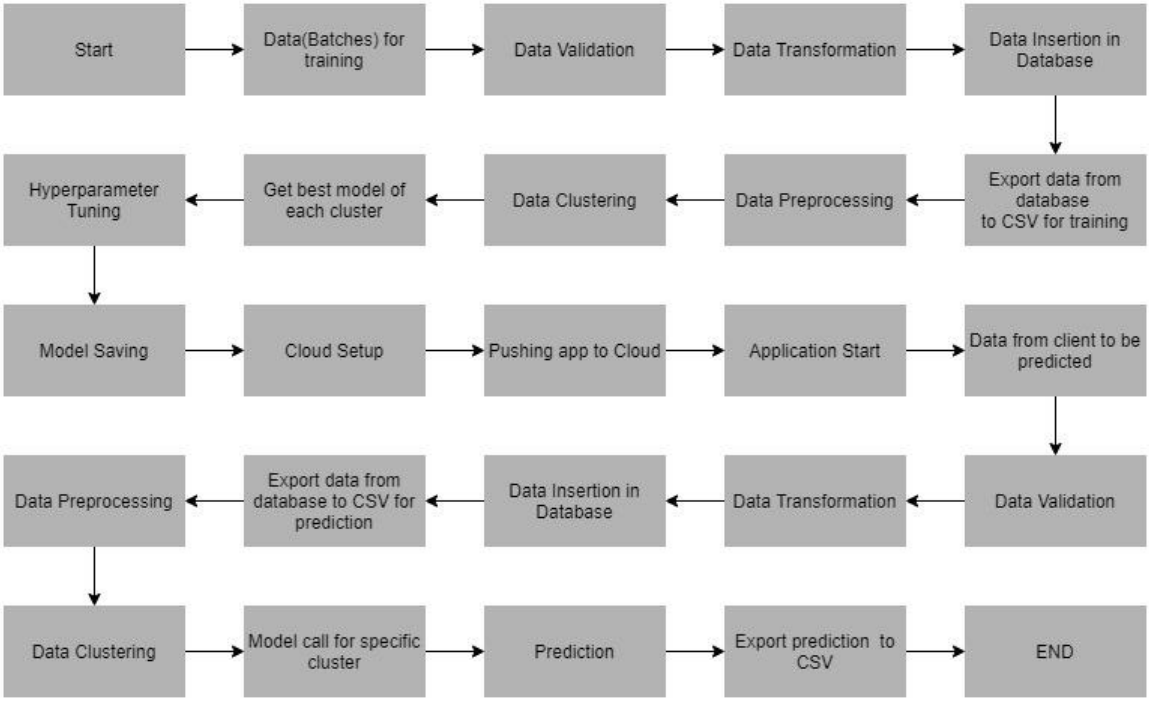
Problem Statement

To build a regression model to predict the cost of restaurant for two people based on the given indicators in the training data.

Architecture



Data Description

Data Description: This dataset predicts the visibility distance based on the different indicators as below:

1. serial: The index/serial number
2. URL : The URL for the restaurant
3. address: The address of the restaurant.
4. name: The name of the restaurant.
5. online_order: Does the restaurant allow online order(Yes/No)
6. book_table: Does the restaurant allow table booking(Yes/No)
7. rate: Restaurant rating out of 5.
8. votes: The number of votes for the restaurant.
9. Phone: Resyaurant contact number
10. Location: Location of the restaurant
11. rest_type:Type of the restaurant(Casual Dining, Café etc.).
12. dish_liked: The Most liked dishes in the restaurant(Pasta, Lunch Buffet, Masala Papad etc.)

```
13. cuisines: North Indian, Mughlai, Chinese etc.
14. approx_cost(for two people): In Rupees. (target Column)
15. reviews_list: The List of reviews
16. menu_item: Open Dosa', 'Benne Set Dosa' etc.
17. listed_in(type): Buffet, café etc
18. listed_in(city): The part of city where restaurant is listed.
```

Apart from training files, we also require a "schema" file from the client, which contains all the relevant information about the training files such as:

Name of the files, Length of Date value in FileName, Length of Time value in FileName, Number of Columns, Name of the Columns, and their datatype.

Data Validation

In this step, we perform different sets of validation on the given set of training files.

1. Name Validation- We validate the name of the files based on the given name in the schema file. We have created a regex pattern as per the name given in the schema file to use for validation. After validating the pattern in the name, we check for the length of date in the file name as well as the length of time in the file name. If all the values are as per requirement, we move such files to "Good_Data_Folder" else we move such files to "Bad_Data_Folder."

2. Number of Columns - We validate the number of columns present in the files, and if it doesn't match with the value given in the schema file, then the file is moved to "Bad_Data_Folder."

3. Name of Columns - The name of the columns is validated and should be the same as given in the schema file. If not, then the file is moved to "Bad_Data_Folder".

4. The datatype of columns - The datatype of columns is given in the schema file. This is validated when we insert the files into Database. If the datatype is wrong, then the file is moved to "Bad_Data_Folder".

5. Null values in columns - If any of the columns in a file have all the values as NULL or missing, we discard such a file and move it to "Bad_Data_Folder".

Data Insertion in Database

1) Database Creation and connection - Create a database with the given name passed. If the database is already created, open the connection to the database.

2) Table creation in the database - Table with name - "Good_Data", is created in the database for inserting the files in the "Good_Data_Folder" based on given column names and datatype in the schema file. If the table is already present, then the new table is not created and new files are inserted in the already present table as we want training to be done on new as well as old training files.

3) Insertion of files in the table - All the files in the "Good_Data_Folder" are inserted in the above-created table. If any file has invalid data type in any of the columns, the file is not loaded in the table and is moved to "Bad_Data_Folder".

Model Training

1) Data Export from Db - The data in a stored database is exported as a CSV file to be used for model training.

2) Data Preprocessing

a) Drop columns not useful for training the model. Such columns were selected while doing the EDA.

b) Replace the invalid values with numpy “nan” so we can use imputer on such values.

d) Check for null values in the columns. If present, remove them.

e) Scale the training and test data separately

f) Encode the categorical values.

3) Clustering - KMeans algorithm is used to create clusters in the preprocessed data. The optimum number of clusters is selected by plotting the elbow plot, and for the dynamic selection of the number of clusters, we are using "KneeLocator" function. The idea behind clustering is to implement different algorithms

To train data in different clusters. The Kmeans model is trained over preprocessed data and the model is saved for further use in prediction.

4) Model Selection - After clusters are created, we find the best model for each cluster. We are using two algorithms, "Decision Tree Regressor" and "XGBoost regressor". For each cluster, both the algorithms are passed with the best parameters derived from GridSearch. We calculate the Rsquared scores for both models and select the model with the best score. Similarly, the model is selected for each cluster. All the models for every cluster are saved for use in prediction.

Prediction Data Description

Client will send the data in multiple set of files in batches at a given location. Data will contain climate indicators in 10 columns.

Apart from prediction files, we also require a "schema" file from client which contains all the relevant information about the training files such as:

Name of the files, Length of Date value in FileName, Length of Time value in FileName, Number of Columns, Name of the Columns and their datatype.

Data Validation

In this step, we perform different sets of validation on the given set of training files.

1) Name Validation- We validate the name of the files on the basis of given Name in the schema file. We have created a regex pattern as per the name given in schema file, to use for validation. After validating the pattern in the name, we check for length of date in the file name as well as length of time in the file name. If all the values are as per requirement, we move such files to "Good_Data_Folder" else we move such files to "Bad_Data_Folder".

2) Number of Columns - We validate the number of columns present in the files, if it doesn't match with the value given in the schema file then the file is moved to "Bad_Data_Folder".

3) Name of Columns - The name of the columns is validated and should be same as given in the schema file. If not, then the file is moved to "Bad_Data_Folder".

4) Datatype of columns - The datatype of columns is given in the schema file. This is validated when we insert the files into Database. If dataype is wrong then the file is moved to "Bad_Data_Folder".

5) Null values in columns - If any of the columns in a file has all the values as NULL or missing, we discard such file and move it to "Bad_Data_Folder".

Data Insertion in Database

1) Database Creation and connection - Create database with the given name passed. If the database is already created, open the connection to the database.

2) Table creation in the database - Table with name - "Good_Data", is created in the database for inserting the files in the "Good_Data_Folder" on the basis of given column names and datatype in the schema file. If table is already present then new table is not created, and new files are inserted the already present table as we want training to be done on new as well old training files.

3) Insertion of files in the table - All the files in the "Good_Data_Folder" are inserted in the above-created table. If any file has invalid data type in any of the columns, the file is not loaded in the table and is moved to "Bad_Data_Folder".

Prediction

1) Data Export from Db - The data in the stored database is exported as a CSV file to be used for prediction.

2) Data Preprocessing

a) Drop columns not useful for training the model. Such columns were selected while doing the EDA.

b) Replace the invalid values with numpy “nan” so remove them.

d) Scale the training data.

e) convert categorical to numerical

3) Clustering - KMeans model created during training is loaded, and clusters for the preprocessed prediction data is predicted.

4) Prediction - Based on the cluster number, the respective model is loaded and is used to predict the data for that cluster.

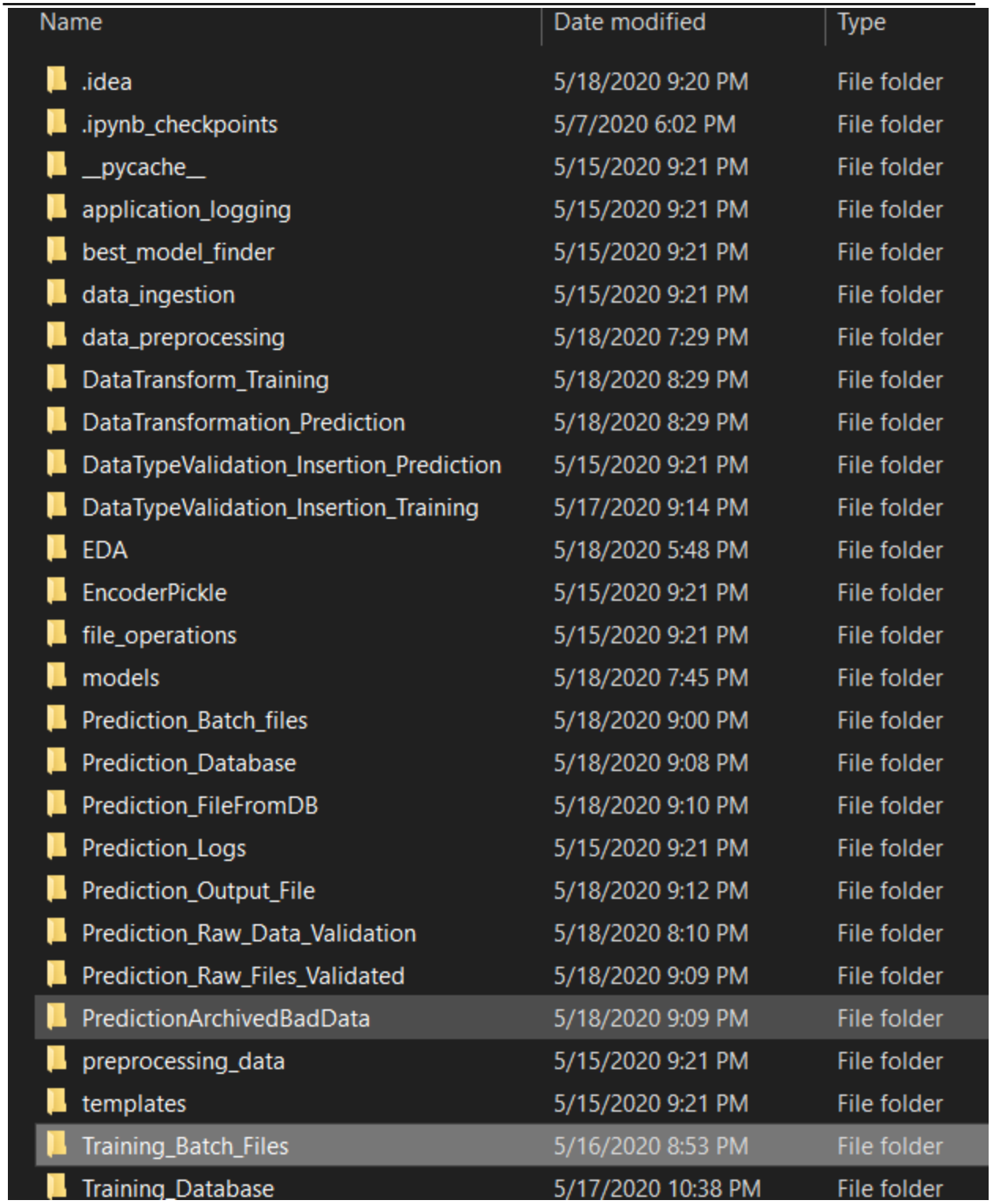
5) Once the prediction is made for all the clusters, the predictions along with the original names before label encoder are saved in a CSV file at a given location and the location is returned to the client.

Deployment

We will be deploying the model to the Pivotal Web Services Platform.

This is a workflow diagram for the prediction of using the trained model.

Now let’s see the zomato project folder structure.



Name	Date modified	Type
.idea	5/18/2020 9:20 PM	File folder
.ipynb_checkpoints	5/7/2020 6:02 PM	File folder
__pycache__	5/15/2020 9:21 PM	File folder
application_logging	5/15/2020 9:21 PM	File folder
best_model_finder	5/15/2020 9:21 PM	File folder
data_ingestion	5/15/2020 9:21 PM	File folder
data_preprocessing	5/18/2020 7:29 PM	File folder
DataTransform_Training	5/18/2020 8:29 PM	File folder
DataTransformation_Prediction	5/18/2020 8:29 PM	File folder
DataTypeValidation_Insertion_Prediction	5/15/2020 9:21 PM	File folder
DataTypeValidation_Insertion_Training	5/17/2020 9:14 PM	File folder
EDA	5/18/2020 5:48 PM	File folder
EncoderPickle	5/15/2020 9:21 PM	File folder
file_operations	5/15/2020 9:21 PM	File folder
models	5/18/2020 7:45 PM	File folder
Prediction_Batch_files	5/18/2020 9:00 PM	File folder
Prediction_Database	5/18/2020 9:08 PM	File folder
Prediction_FileFromDB	5/18/2020 9:10 PM	File folder
Prediction_Logs	5/15/2020 9:21 PM	File folder
Prediction_Output_File	5/18/2020 9:12 PM	File folder
Prediction_Raw_Data_Validation	5/18/2020 8:10 PM	File folder
Prediction_Raw_Files_Validated	5/18/2020 9:09 PM	File folder
PredictionArchivedBadData	5/18/2020 9:09 PM	File folder
preprocessing_data	5/15/2020 9:21 PM	File folder
templates	5/15/2020 9:21 PM	File folder
Training_Batch_Files	5/16/2020 8:53 PM	File folder
Training_Database	5/17/2020 10:38 PM	File folder

requirements.txt file consists of all the packages that you need to deploy the app in the cloud.

```
def predictRouteClient():
try:
    if request.json is not None:
        path = request.json['filepath']

        pred_val = pred_validation(path) #object initialization

        pred_val.prediction_validation() #calling the prediction validation function

        pred = prediction(path) #object initialization

        # predicting for dataset present in database
        path = pred.predictionFromModel()
        return Response("Prediction File created at %s!!!" % path)
    elif request.form is not None:
        path = request.form['filepath']

        pred_val = pred_validation(path) #object initialization

        pred_val.prediction_validation() #calling the prediction validation function

        pred = prediction(path) #object initialization

        # predicting for dataset present in database
        path = pred.predictionFromModel()
        return Response("Prediction File created at %s!!!" % path)
```

main.py is the entry point of our application, where the flask server starts.

```
    self.log_writer = logger.App_Logger()
    self.pred_data_val = Prediction_Data_validation(path)

def predictionFromModel(self):

    try:
        self.pred_data_val.deletePredictionFile() #deletes the existing prediction file from last run!
        self.log_writer.log(self.file_object,'Start of Prediction')
        data_getter=data_loader_prediction.Data_Getter_Pred(self.file_object,self.log_writer)
        data=data_getter.get_data()

        #code change
        # wafer_names=data['Wafer']
        # data=data.drop(labels=['Wafer'],axis=1)

        preprocessor=preprocessing.Preprocessor(self.file_object,self.log_writer)
        data = preprocessor.dropUnnecessaryColumns(data,['DATE','Precip','WETBULBTEMPF','DewPointTempF','StationPressure'])
```

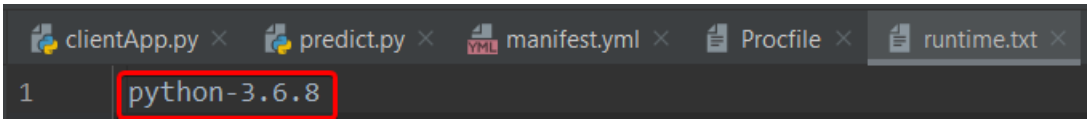
This is the **predictionFromModel.py** file where the predictions take place based on the data we are giving input to the model.

```
---
applications:
- name: zomato
  memory: 2GB
  disk_quota: 1GB
  random-route: true
  parameters:
    memory: 2GB
    buildpack: 'python_buildpack'
```

manifest.yml:- This file contains the instance configuration, app name, and build pack language.

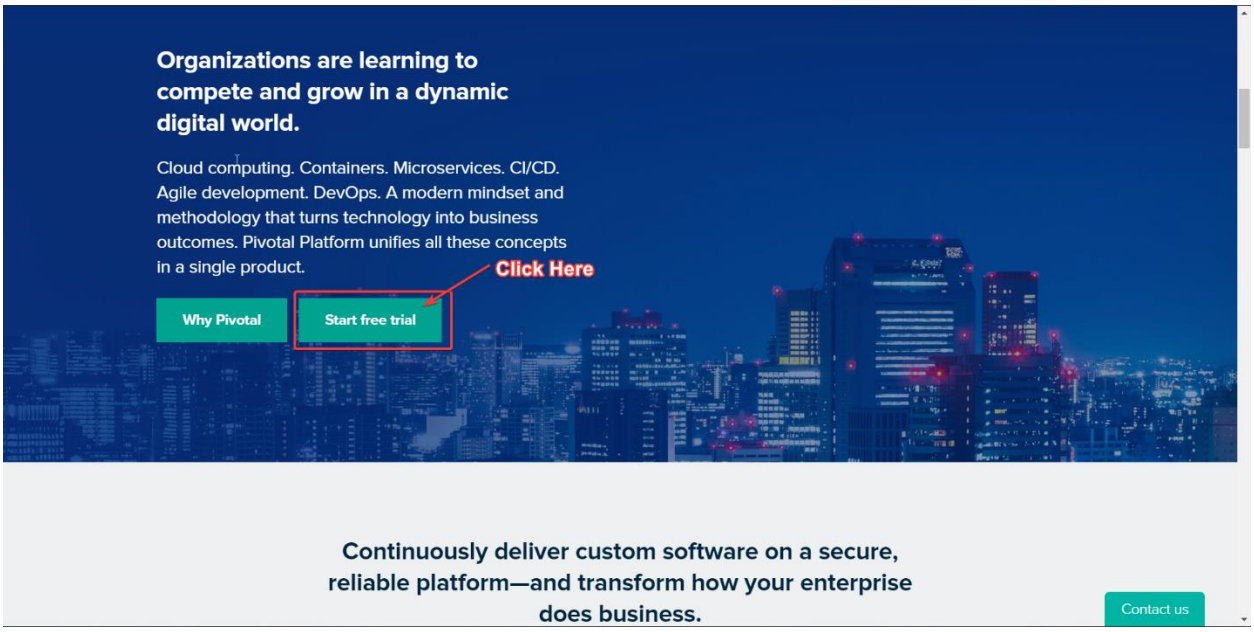
```
main.py × predictFromModel.py × Procfile ×
web: python main.py   --master --processes 4 --threads 2
```

Procfile :- It contains the entry point of the app.

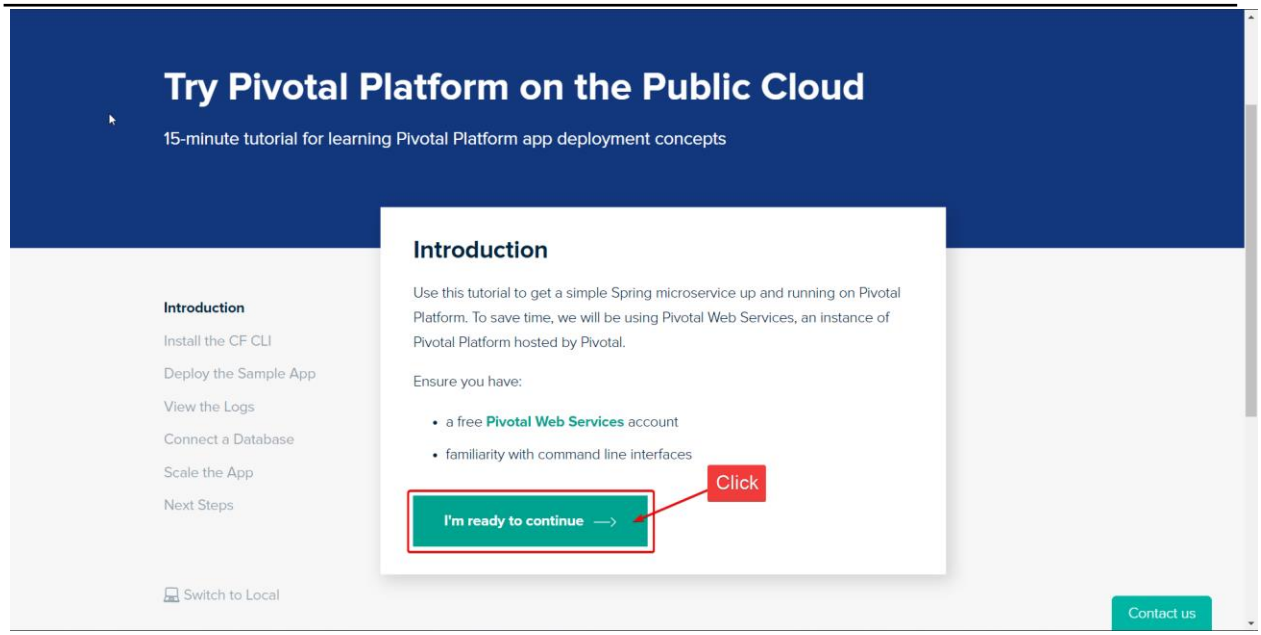


runtime.txt:- It contains the Python version number.

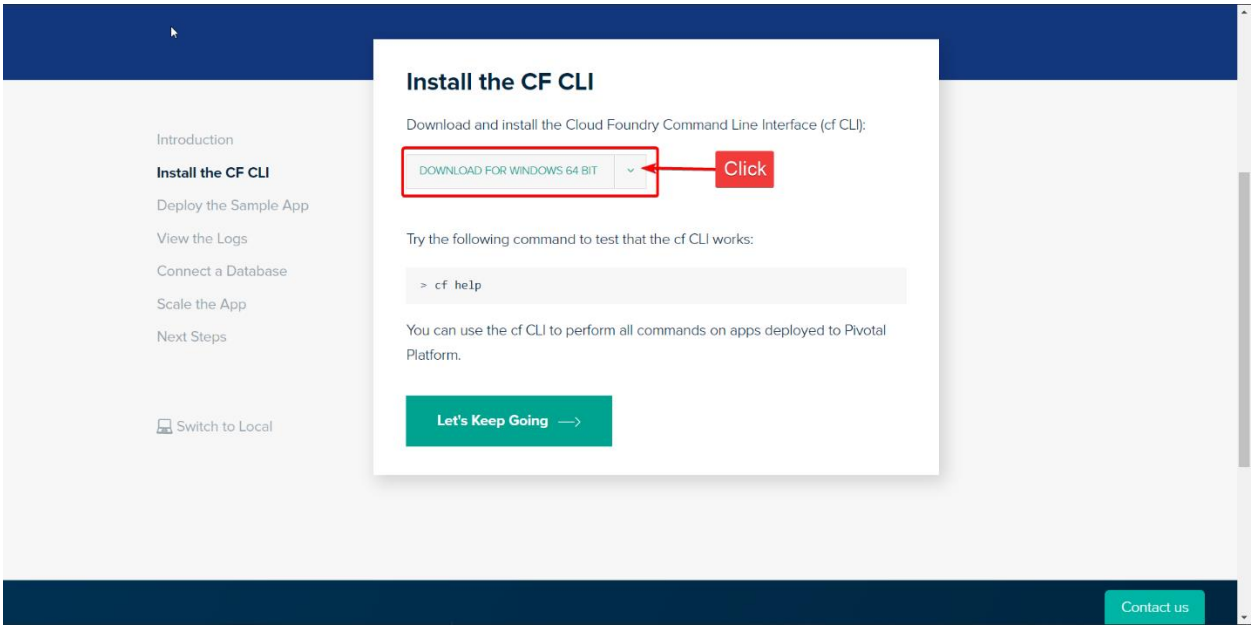
Visit the official website **https://pivotal.io/platform.**



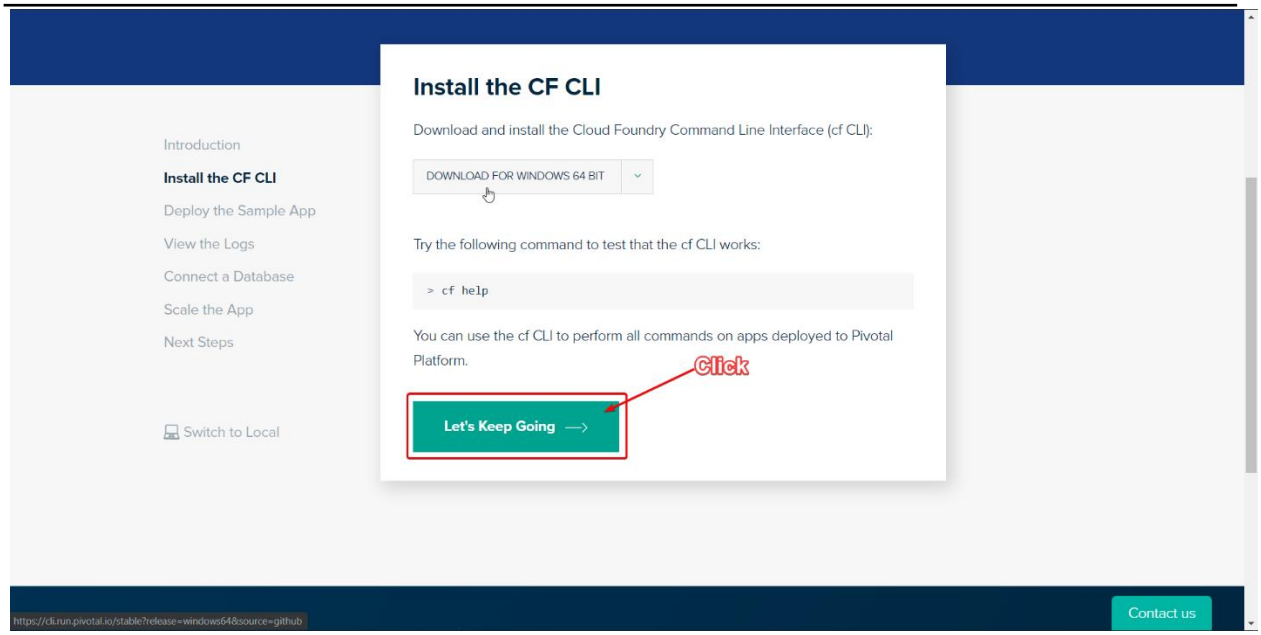
Scroll Down to see the **Start Trial Button**



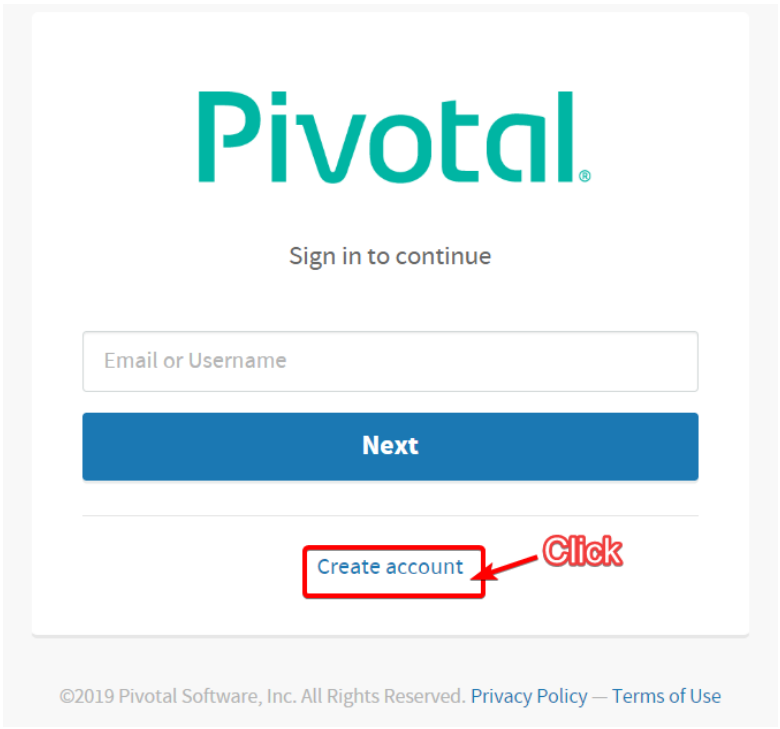
Click on the start trial button and the next interface will open. Then I will click on **I'm ready to continue**



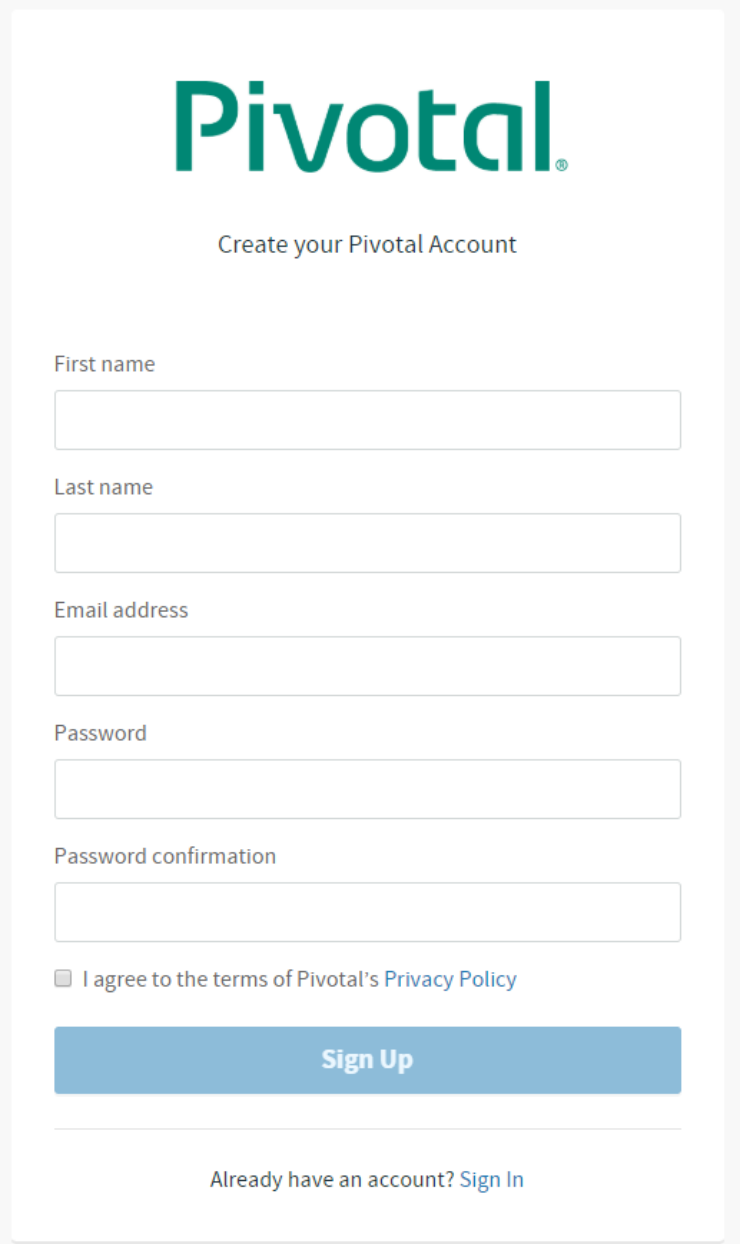
Click on Download for **Windows 64 bit, and** then zip file will be downloaded. Keep it for future uses.



Now click on **Let's Keep Going**



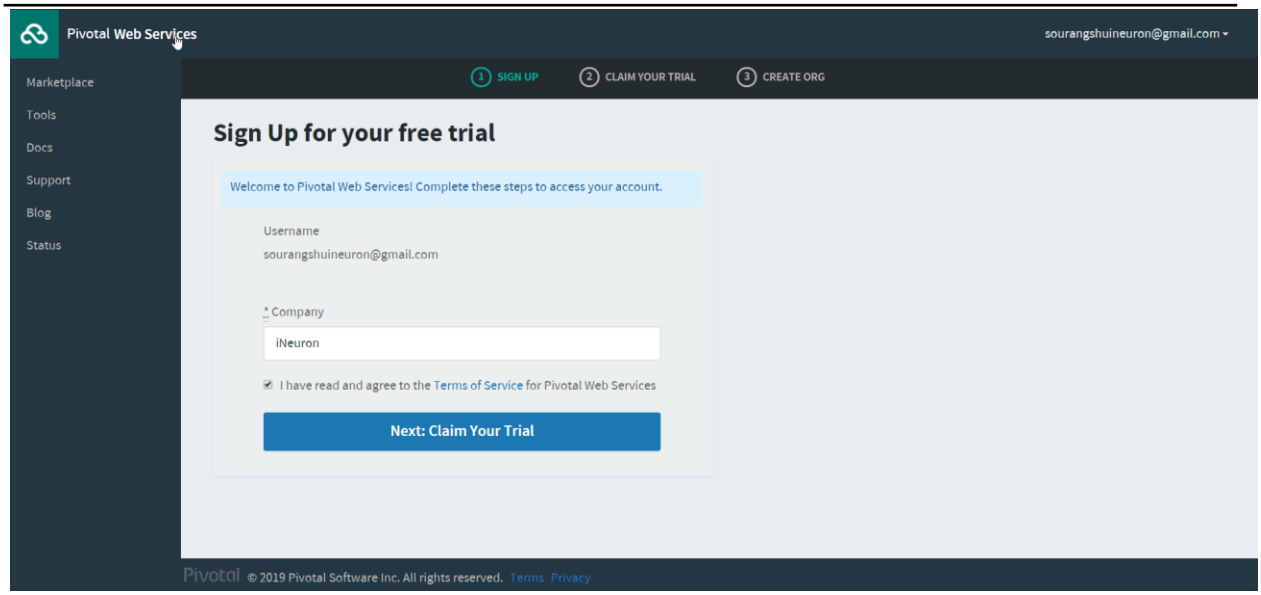
Click on **Create Your Account**



Fill Up your Details For registration

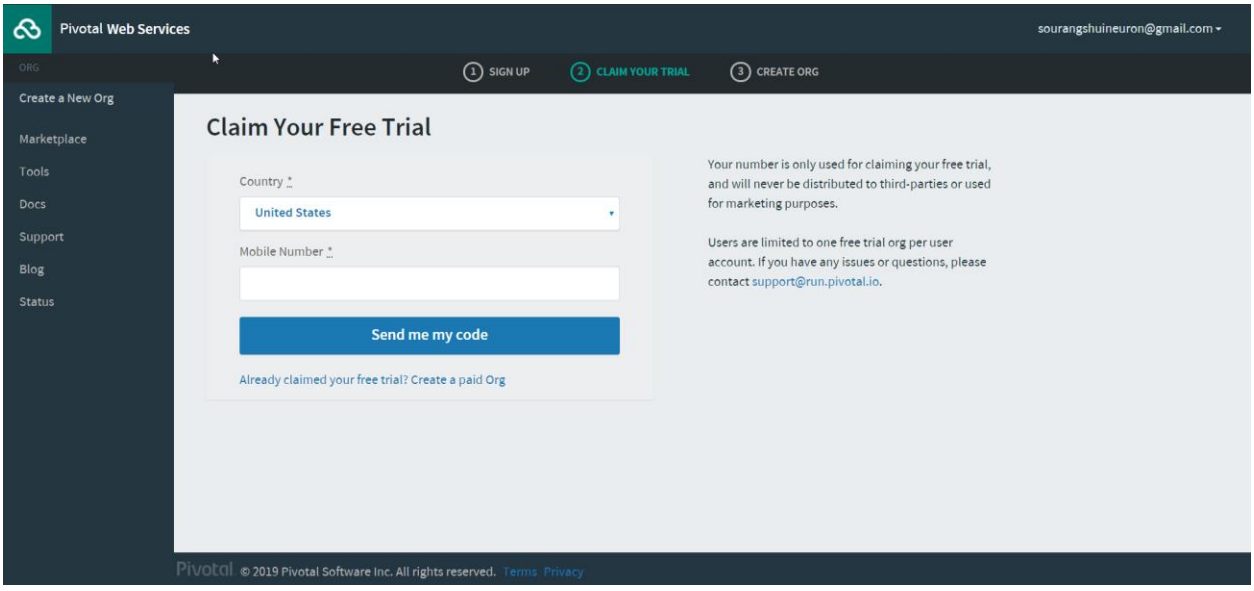
Do the email verification

Then login in again

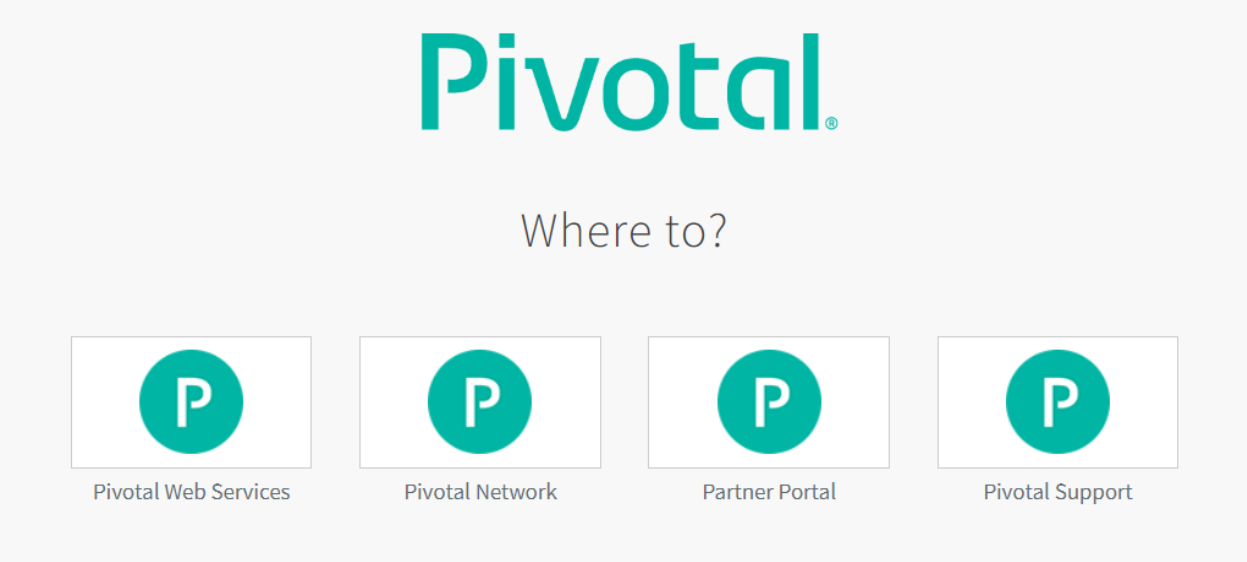


After logging you will see this screen below and start your free trial.

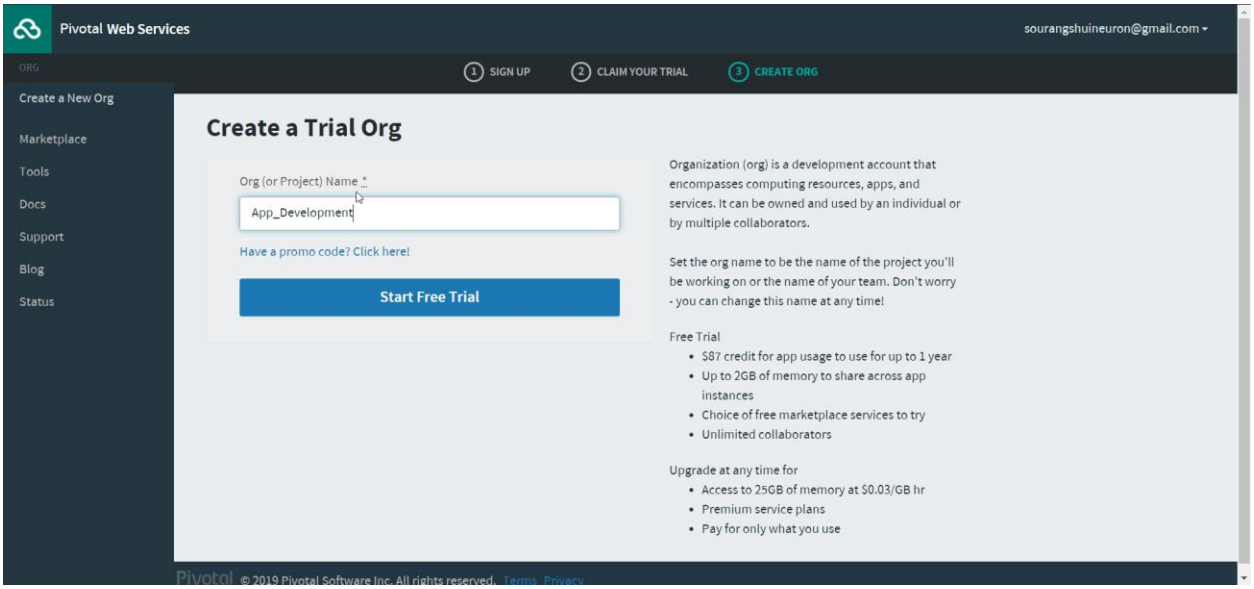
Write any Company or which one you prefer



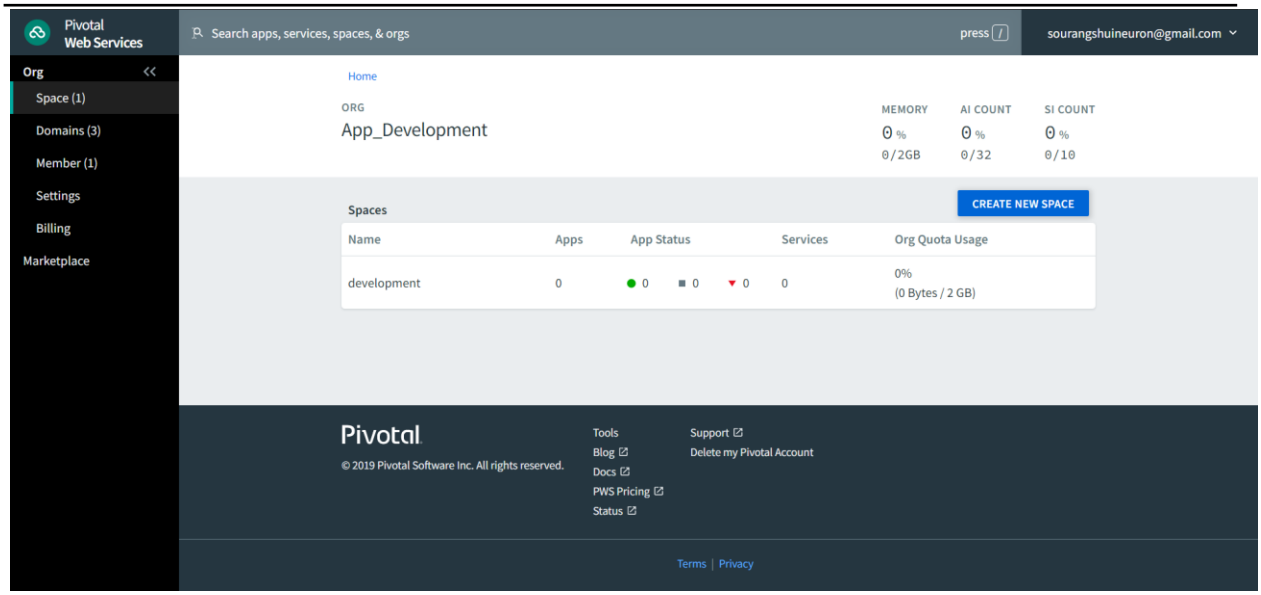
Enter your **Mobile Number** for Verification



Click on **Pivotal Web Services**



Give any **Org** name



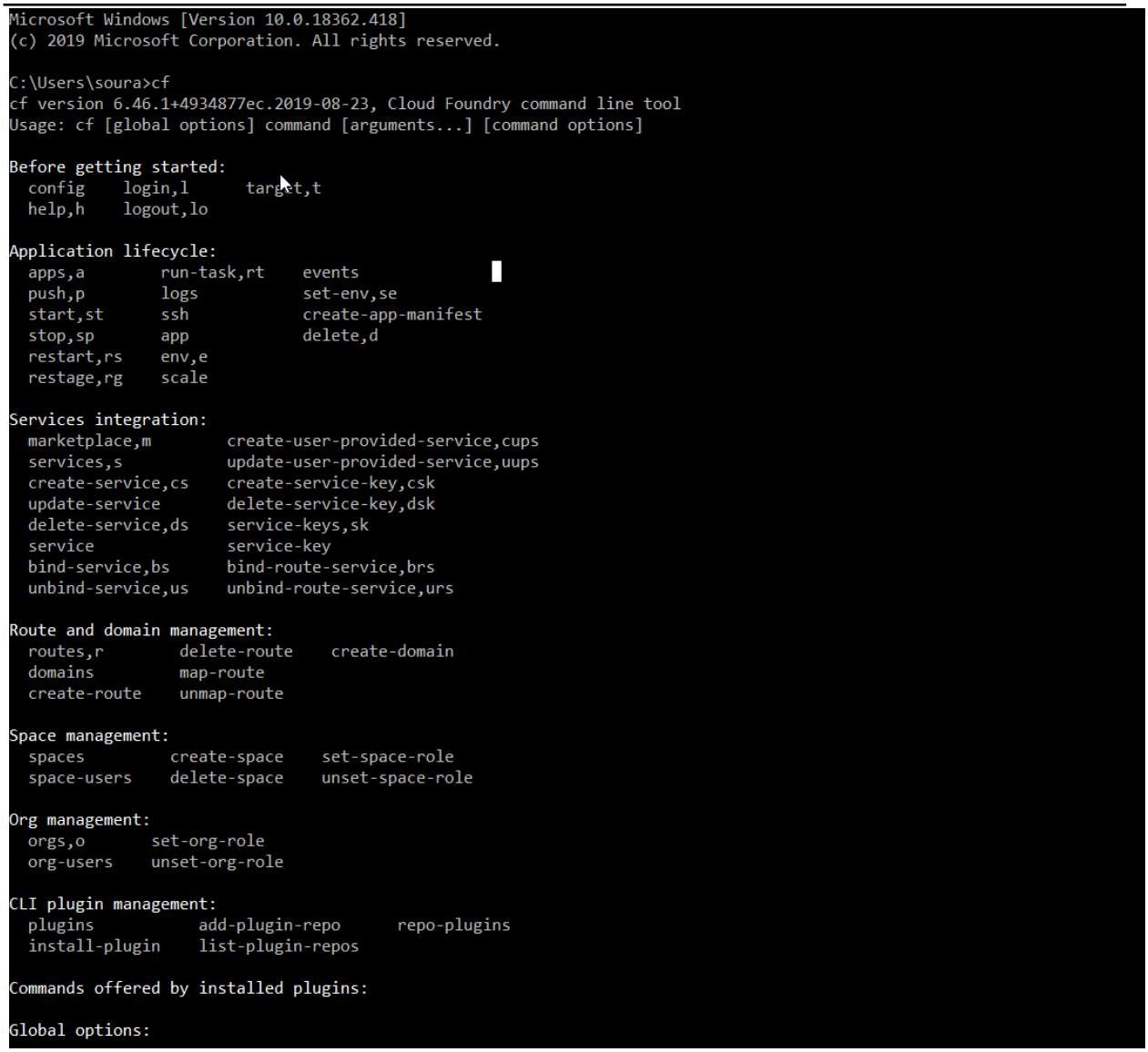
Now you are inside your Org, and by default, development space is created in your org. You can push your apps here.

The cloud signup process is done, and the setup is ready for us to push the app.

Previously you have downloaded the **CLI.zip** file. Unzip the file and install the .exe file with admin rights.

After a successful installation, you can verify by opening your CMD and type **cf**.

Then you will get a screen which is shown below



If you see this screen in your CMD, the installation is successful.

Now type the command to login via cf-cli

cf login -a https://api.run.pivotal.io

Next, enter your email and password. Now you are ready to push your app.

Now let’s go to the app which we have built.

```
Microsoft Windows [Version 10.0.18363.535]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\soura>cf login
API endpoint: https://api.run.pivotal.io

Email> sourangshuineuron@gmail.com

Password>
Authenticating...
OK

Targeted org App_Development

Targeted space development



API endpoint:   https://api.run.pivotal.io (API version: 2.144.0)
User:           sourangshuineuron@gmail.com
Org:            App_Development
Space:          development
```

Navigate to the project folder after downloading.

Then write cf push in the terminal.

After the app is successfully deployed in the cloud, you will see the screen below with the route.

```
   Uploading droplet, build artifacts cache...
   Uploading build artifacts cache...
   Uploading droplet...
   Uploaded build artifacts cache (218.3M)
   Uploaded droplet (291.6M)
   Uploading complete
   Cell 732975b5-a95c-4e37-b595-a0a3c3a9e2ea stopping instance 9f24f6ff-be7a-4eda-a6b3-81cb3c8bb315
   Cell 732975b5-a95c-4e37-b595-a0a3c3a9e2ea destroying container for instance 9f24f6ff-be7a-4eda-a6b3-81cb3c8bb315
   Cell 732975b5-a95c-4e37-b595-a0a3c3a9e2ea successfully destroyed container for instance 9f24f6ff-be7a-4eda-a6b3-81cb3c8bb315

Waiting for app to start...

name:              WaferQuality
requested state:   started
routes:            waferquality-fearless-topi-vk.cfapps.io
last uploaded:     Fri 31 Jan 21:01:17 IST 2020
stack:             cflinuxfs3
buildpacks:        python

type:            web
instances:       1/1
memory usage:    2048M
start command:   python main.py    --master --processes 4 --threads 2
     state     since                  cpu     memory        disk          details
#0   running   2020-01-31T15:32:10Z   20.1%   55.9M of 2G   800.8M of 2G


D:\Project_\WaferFaultDetection>

D:\Project_\WaferFaultDetection>
```

Api Route

Finally, the app is pushed in the cloud.

Lets Open Postman and see the result.